ENGENHARIA DE SOFTWARE
INTRODUÇÃO

Luís Morgado

2023

ENGENHARIA DE SOFTWARE

- A *engenharia de software* é uma área de engenharia orientada para a especificação, desenvolvimento e manutenção de software, que tem por objectivo o desenvolvimento, operação e manutenção de software de modo *sistemático* e *quantificável*

- **Sistemático**
  - Significa a capacidade de realizar o desenvolvimento de software de forma organizada e previsível, de modo a garantir a satisfação dos requisitos definidos, incluindo tempo e recursos necessários

- **Quantificável**
  - Significa a capacidade de avaliar os meios envolvidos e os resultados produzidos no desenvolvimento de software, utilizando métodos e processos adequados para garantir a qualidade e o desempenho do software produzido, bem como a criação de documentação e a monitorização do processo de desenvolvimento
  - O desenvolvimento de software de forma quantificável é importante para garantir que os requisitos do software sejam cumpridos, bem como para prever e gerir os recursos necessários para o desenvolvimento e operação do software produzido

ENGENHARIA DE SOFTWARE

- Com o crescimento da utilização de sistemas baseados em software, que vão desde aplicações informáticas e sistemas de informação, a sistemas de controlo industrial ou de veículos autónomos, dois problemas principais têm relevância crescente no desenvolvimento, operação e manutenção de software, *complexidade* e *mudança*
- **Complexidade**
  - A complexidade é expressa em concreto na crescente dificuldade de desenvolvimento, operação e manutenção de software, na crescente sofisticação do software produzido, bem como na quantidade crescente de recursos computacionais envolvidos, nomeadamente, em termos de capacidade de processamento e de memória utilizada
- **Mudança**
  - A mudança é expressa no ritmo crescente a que o software necessita de ser produzido, ou modificado, para satisfazer as necessidades dos respetivos contextos de utilização, quer a nível de funcionalidades disponibilizadas, quer a nível das tecnologias utilizadas

CRESCIMENTO EXPONENCIAL DE RECURSOS COMPUTACIONAIS

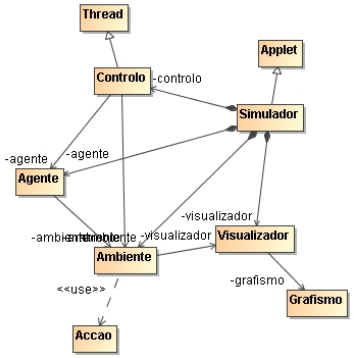
IBM Real-Time Computer Complex - NASA Manned Spacecraft Center
Década de 1960

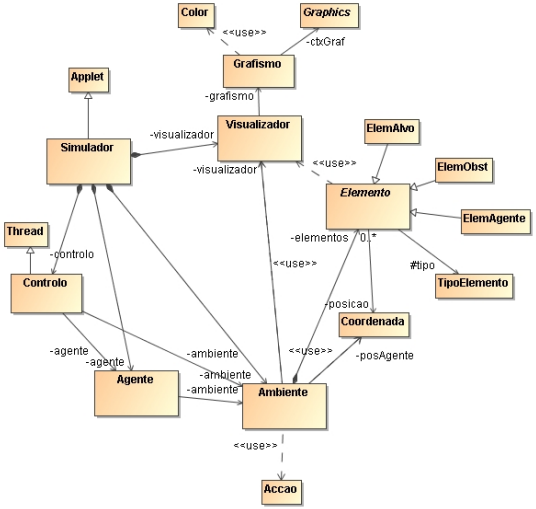
Como exemplo de comparação do crescimento da capacidade computacional disponível nos dispositivos actuais, muitos dos telefones móveis de hoje têm mais capacidade computacional que os computadores envolvidos na missão lunar da NASA na década de 1960

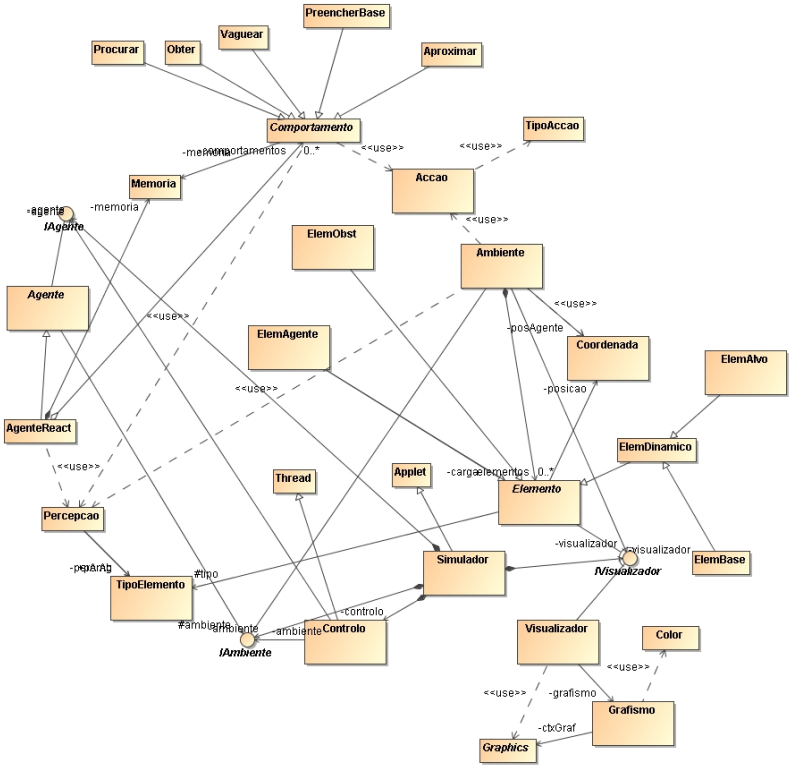
SOFTWARE

Hoje

SOFTWARE E COMPLEXIDADE







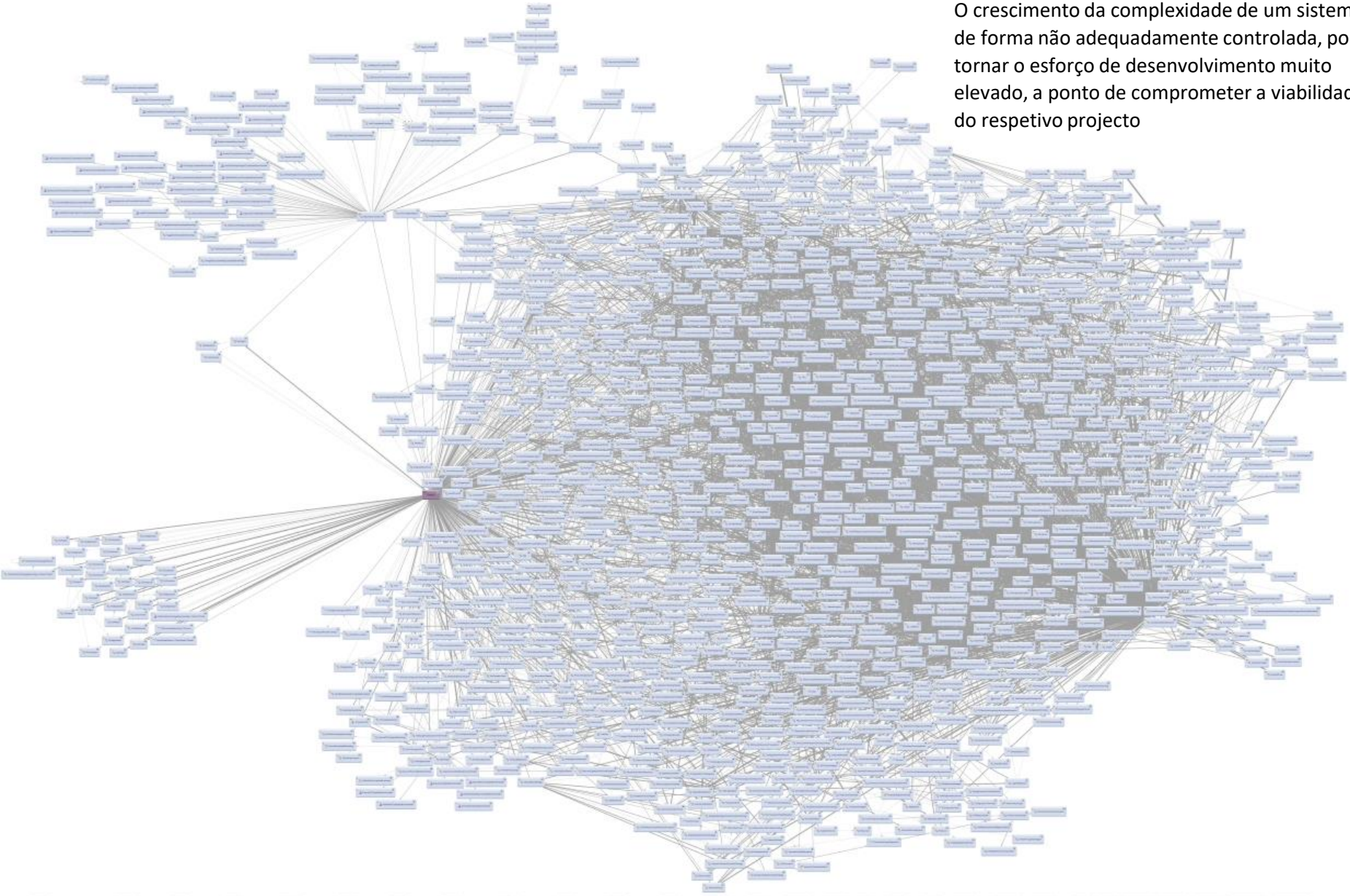
Em que consiste a complexidade quando estamos a falar do desenvolvimento de software?

Numa primeira perspetiva, a complexidade consiste numa dificuldade crescente em compreender e gerir as partes e as relações entre partes que constituem um sistema de software, a qual se pode observar numa representação gráfica (num modelo) desse software.

Na prática, essa complexidade traduz-se na dificuldade e esforço crescente para a concepção e implementação do software, à medida que vão sendo incluídos mais aspectos do seu funcionamento.

SOFTWARE E COMPLEXIDADE

O crescimento da complexidade de um sistema de forma não adequadamente controlada, pode tornar o esforço de desenvolvimento muito elevado, a ponto de comprometer a viabilidade do respetivo projecto



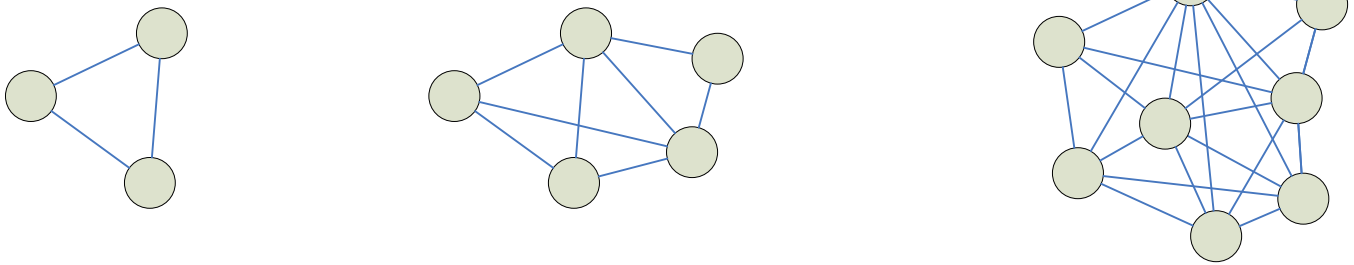
COMPLEXIDADE

Grau de **dificuldade de previsão** das propriedades de um sistema dadas as propriedades das partes individuais [Weaver, 1948]

- Relacionada com a **informação** que é necessária para a caracterização de um sistema
- Um sistema é tanto mais **complexo** quanto mais **informação** for necessária para a sua **descrição**
- Reflecte-se no **esforço** necessário para geração da **organização** (**ordem**) do sistema

O PROBLEMA DA COMPLEXIDADE

COMPLEXIDADE ESTRUTURAL



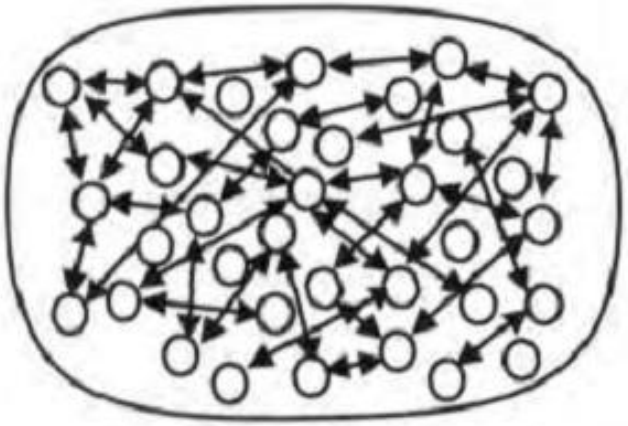
- **UM PROBLEMA DE INTERACÇÃO**
  - De partes do sistema
  - De elementos de informação
  - De elementos das equipas de desenvolvimento
- **EXPLOSÃO COMBINATÓRIA**
  - Um sistema com duas vezes mais partes é muito mais do que duas vezes mais complexo

CRESCIMENTO EXPONENCIAL
DA COMPLEXIDADE

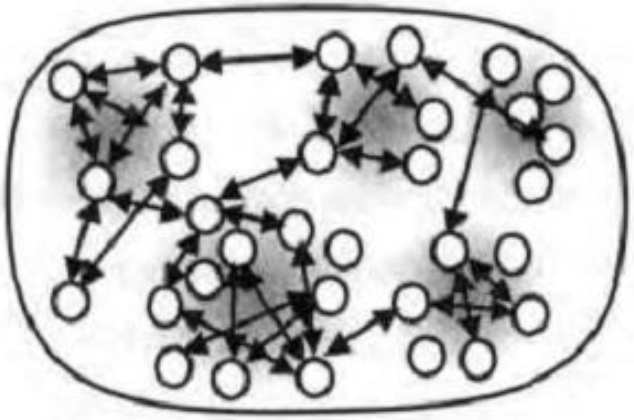
COMPLEXIDADE E ORGANIZAÇÃO

Diferentes tipos de complexidade associados à organização de um sistema

- Ordem, organização, função, propósito
- Desordem, desorganização, perda de função e de propósito



AN UNSTRUCTURED SYSTEM



AN "ORGANIZED" SYSTEM

[Flake, 1998]

TIPOS DE COMPLEXIDADE

COMPLEXIDADE **DESORGANIZADA**

- resulta do **número e heterogeneidade** das partes de um sistema
- as partes podem interactuar entre si, mas a **interacção é irregular**
- as características globais do sistema podem ser inferidas com base em **métodos estatísticos**

COMPLEXIDADE **ORGANIZADA**

- resulta dos padrões de **inter-relacionamento** entre as partes
- as interacções entre partes obedecem a **padrões correlacionáveis** no espaço e no tempo
- **ORDEM, ORGANIZAÇÃO**
  - Propósito (finalidade)
  - Função

ARQUITECTURA DE SOFTWARE

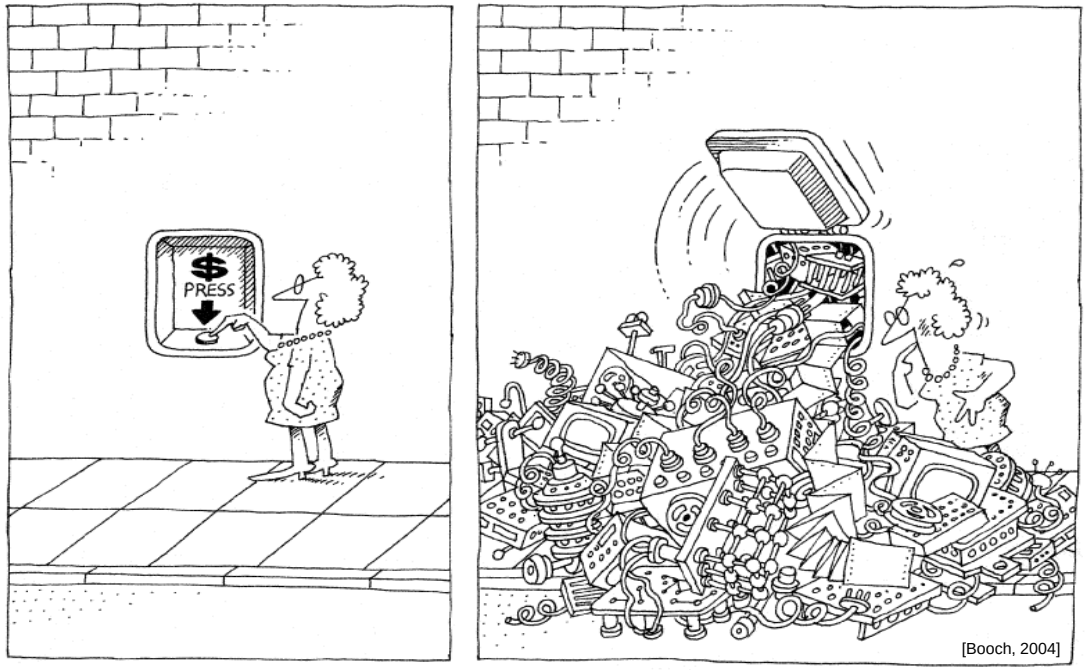
Orientada para abordar o problema da complexidade com base em três vertentes principais: *métricas*, *princípios* e *padrões*

- **MÉTRICAS**
- **PRINCÍPIOS**
- **PADRÕES**

Objectivo: reduzir a complexidade desorganizada e controlar a complexidade organizada necessária à função do sistema

COMPLEXIDADE

- Redução
- Controlo



Quando a complexidade não é adequadamente resolvida, mas apenas acumulada, expressar-se-á em algum momento…

MÉTRICAS DE ARQUITECTURA

Definem *medidas de quantificação* da arquitectura de um software *indicadoras da qualidade* dessa arquitectura

- **ACOPLAMENTO**
  - Grau de interdependência entre subsistemas
- **COESÃO**
  - Nível coerência funcional de um subsistema/módulo (até que ponto esse módulo realiza uma única função)
- **SIMPLICIDADE**
  - Nível de facilidade de compreensão/comunicação da arquitectura
- **ADAPTABILIDADE**
  - Nível de facilidade de alteração da arquitectura para incorporação de novos requisitos ou de alterações nos requisitos previamente definidos

ACOPLAMENTO

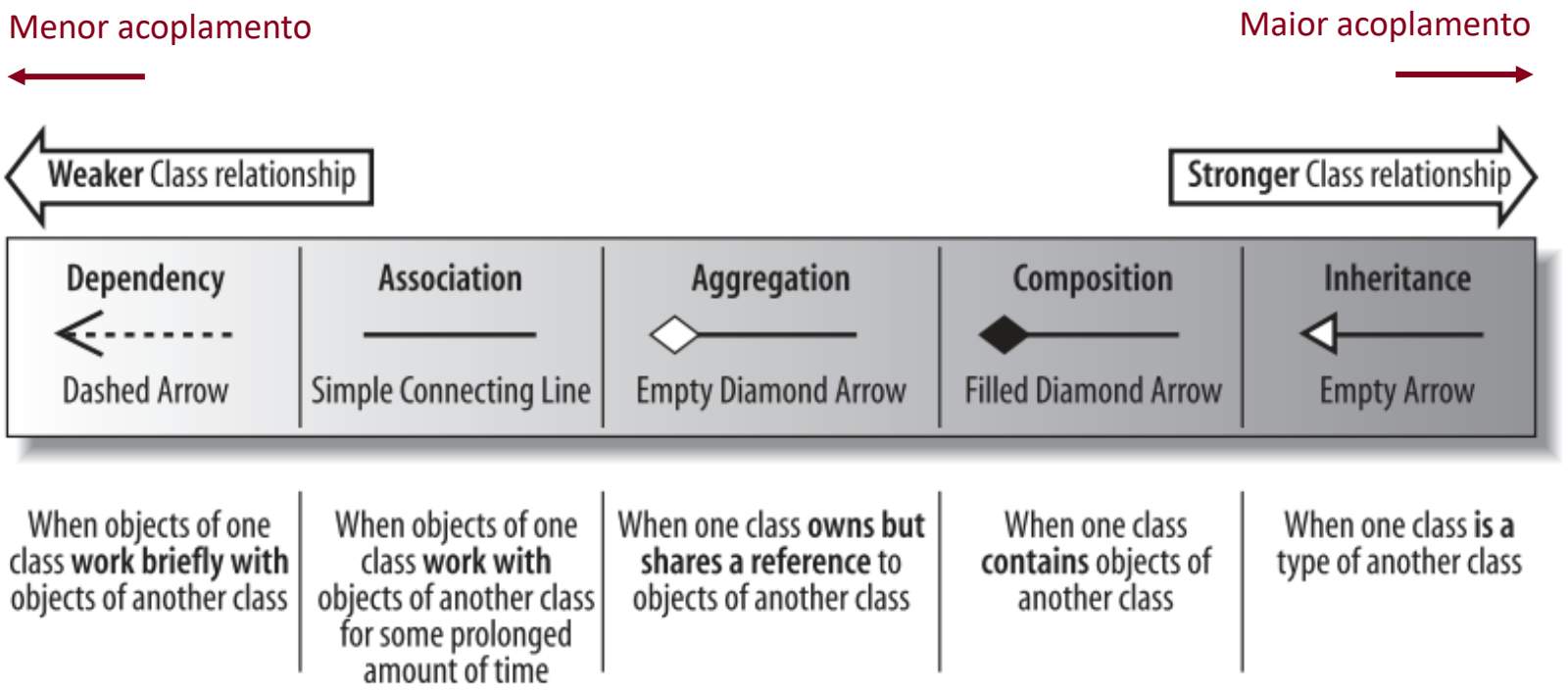
- O conceito de *acoplamento* refere-se ao grau de dependência entre diferentes partes de um sistema, sendo tanto maior quanto maior a dependência entre as partes de um sistema
- O acoplamento pode ser medido utilizando diferentes métricas, como o número de associações entre elementos ou o grau de complexidade de uma interface
- O objetivo é criar software com o menor acoplamento possível, de modo a reduzir a complexidade e a facilitar a sua manutenção e evolução
- A modularidade e o encapsulamento são princípios de arquitectura de software que contribuem para a redução do acoplamento

- **ACOPLAMENTO**
  - Grau de interdependência entre partes de um sistema
  - Característica **inter-modular**
    - Expressa relações entre partes (módulos)
  - Deve ser minimizado
    - Redução de complexidade, facilidade de manutenção e evolução

ACOPLAMENTO EM MODELOS DE ESTRUTURA

Linguagem UML

Relações entre classes e nível de acoplamento



[Miles & Hamilton, 2006]

COESÃO

- O conceito de *coesão* refere-se à forma como os elementos de um sistema estão agrupados de forma coerente entre si, será tanto maior quando mais relacionados entre si forem os elementos agrupados em cada módulo
- A coesão pode ser medida utilizando diferentes critérios, por exemplo, por tipo de função das partes agrupadas, sendo designada neste caso por *coesão funcional*
- O objetivo é criar software que seja claro e fácil de entender, manter e evoluir, facilitando a reutilização e aumentando assim a eficiência do desenvolvimento
- A modularidade e a factorização são princípios de arquitectura de software que contribuem para o aumento da coesão

- **COESÃO**
  - Nível de coerência das partes agrupadas num módulo ou subsistema
    Característica **intra-modular**
    - Expressa relações interiores aos módulos (agrupamentos)
  - Deve ser maximizada
    - Redução de complexidade, facilidade de evolução, manutenção e reutilização

PRINCÍPIOS DE ARQUITECTURA

Definem *meios orientadores* da concepção de arquitectura de software, no sentido de garantir a qualidade da arquitectura produzida, nomeadamente, no que se refere à *minimização do acoplamento*, à *maximização da coesão* e à *gestão da complexidade*

- **MODULARIDADE**
  - **DECOMPOSIÇÃO**
  - **ENCAPSULAMENTO**
- **FACTORIZAÇÃO**
- **ABSTRACÇÃO**

MODULARIDADE

O conceito de *modularidade* refere-se à capacidade de organização de um sistema em partes coesas, ou módulos, que podem ser interligados entre si para produzir a função do sistema, está relacionado com dois aspectos principais, *decomposição* e *encapsulamento*

- **DECOMPOSIÇÃO**
  - De um sistema em partes coesas
    - Para sistematizar interacções
    - Para lidar com a explosão combinatória
  - **FACTORIZAÇÃO**
    - Eliminação de redundância
    - Garantia de consistência
- **ENCAPSULAMENTO**
  - **Isolamento dos detalhes internos** das partes de um sistema em relação ao exterior
    - Para reduzir dependências (interacções)
    - Relacionar estrutura e função no contexto de uma parte
    - Acesso exclusivo através das *interfaces* disponibilizadas
  - **INTERFACES**
    - **Contractos funcionais** para interação com o exterior

FACTORIZAÇÃO

O conceito de *factorização* refere-se à decomposição das partes de um sistema de modo a *eliminar redundância* (partes repetidas), relaciona-se com o conceito matemático correspondente de decomposição de uma expressão em *factores* (partes de um produto), por exemplo, $ax + bx = (a + b)x$

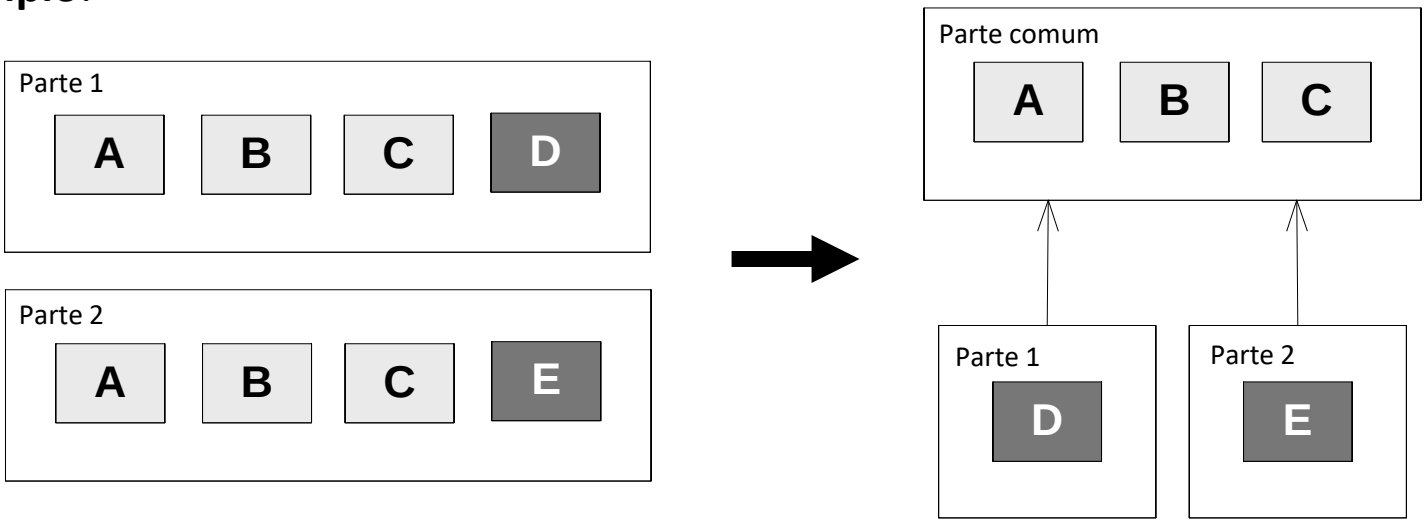
REDUNDÂNCIA

Existência de partes repetidas num sistema, é uma das principais causas de anomalias e de complexidade desorganizada no desenvolvimento de software

Redução de redundância por factorização

As partes repetidas são eliminadas, as partes mantidas são partilhadas

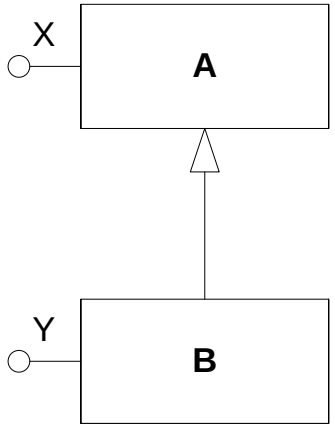
Exemplo:



MECANISMOS DE FACTORIZAÇÃO

HERANÇA

- Factorização estrutural
- B **é** A
- Nível de **acoplamento alto**

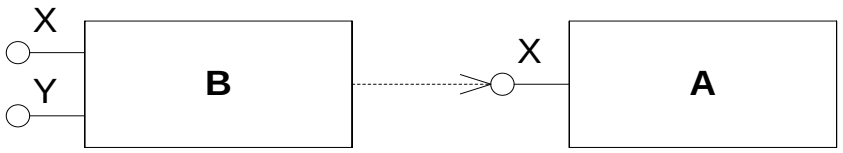


B **é** A

B disponibiliza interfaces X e Y herdando X de A

DELEGAÇÃO

- Factorização funcional
- B **utiliza** A
- Nível de **acoplamento baixo**
- Acoplamento pode variar **dinamicamente**



B **utiliza** A

B disponibiliza interfaces X e Y utilizando A para delegar a funcionalidade de X

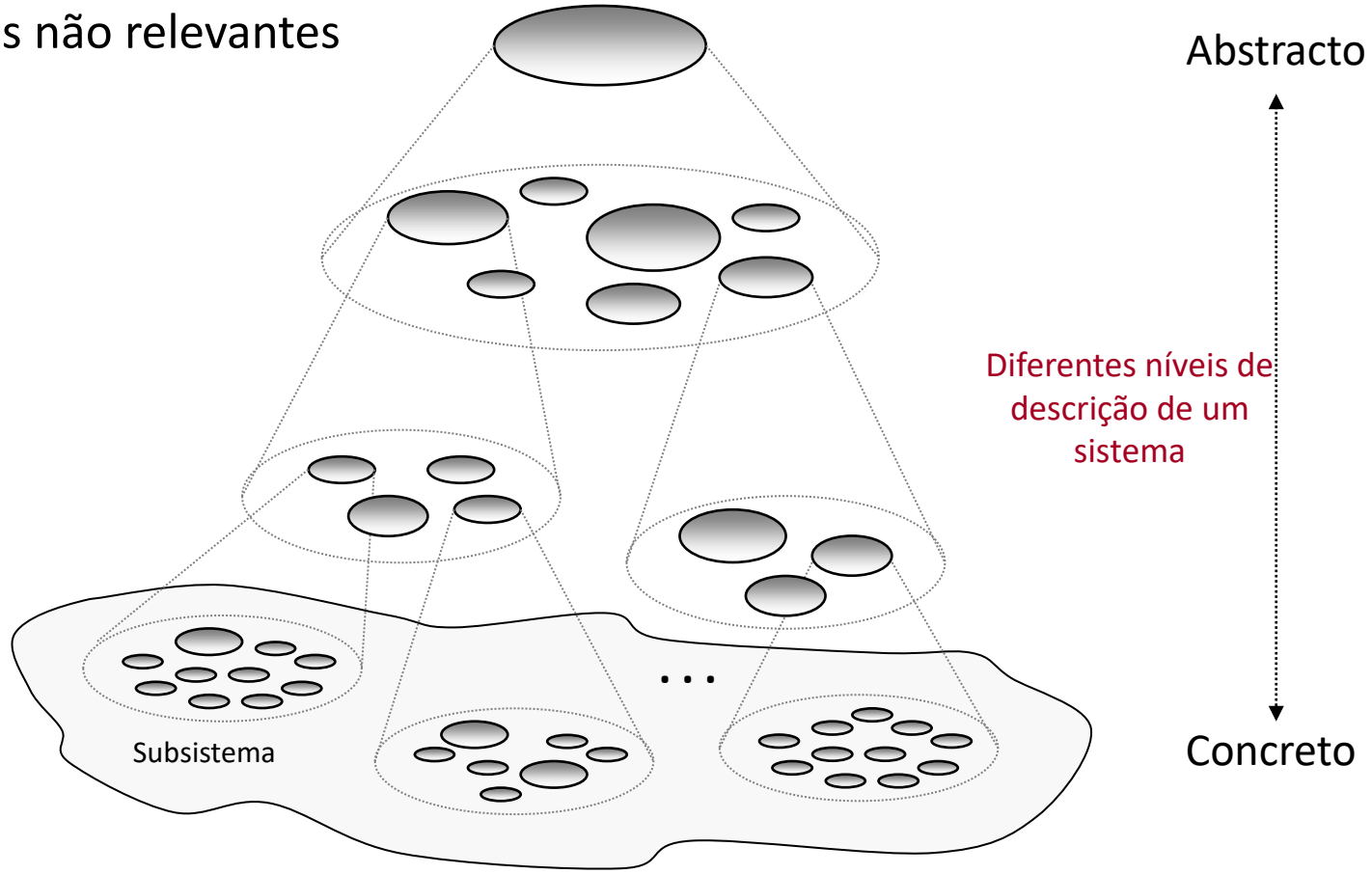
ABSTRACÇÃO

- Processo de descrição de conhecimento a *diferentes níveis de detalhe* (**quantidade de informação**) e *tipos de representação* (**estrutura da informação**) [Korf, 1980]
- **Abstracção é uma ferramenta base para lidar com a complexidade**
  - Identificação de características comuns a diferentes partes
  - Realçar o que é essencial, omitir detalhes não relevantes
  - Modelos
- Desenvolvimento de um sistema complexo
  - Criação de ordem de forma progressiva
  - Processo iterativo guiado por conhecimento

ABSTRACÇÃO

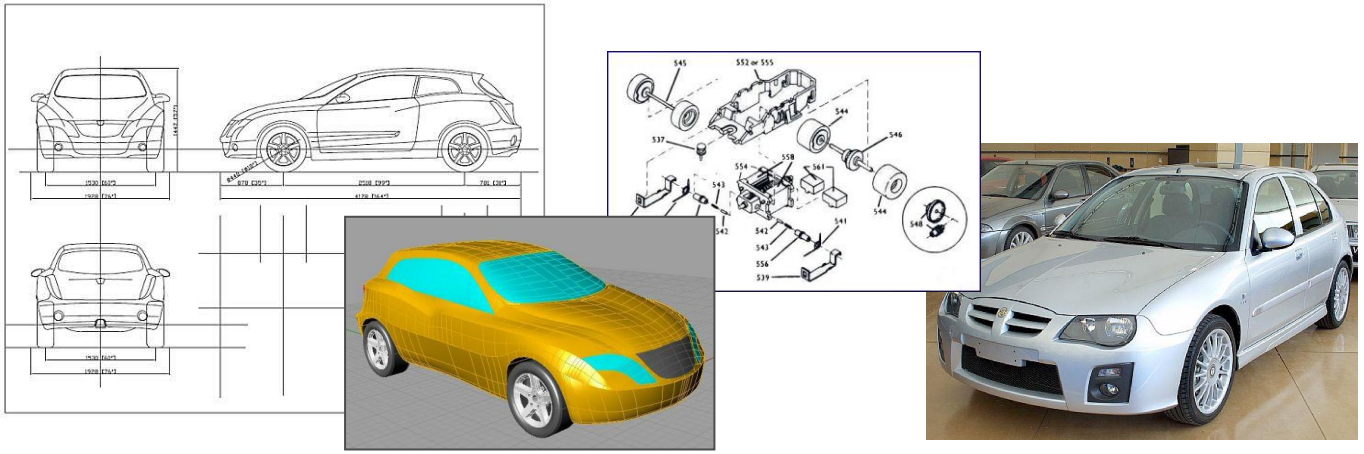
FERRAMENTA BASE PARA LIDAR COM A COMPLEXIDADE

- **Realçar** o que é **essencial**, omitir detalhes não relevantes
- **Simplificação**
- **Focagem**
- **Modelos**



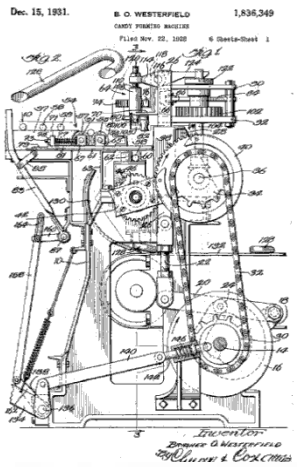
MODELO

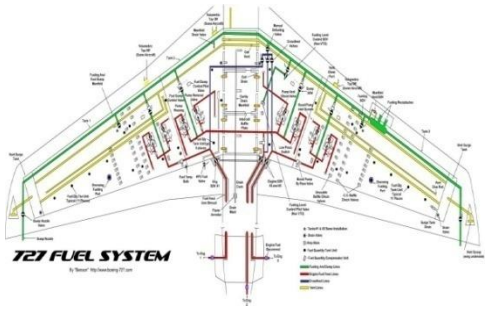
- **Representação abstracta de um sistema**
  - Especificação com base em conceitos abstractos das características fundamentais de um sistema
  - Representação de conhecimento acerca de um sistema
- **Meio para lidar com a complexidade**
  - Obtenção e sistematização progressiva de conhecimento
  - Compreensão e comunicação acerca do sistema
  - Especificação de referência para a realização do sistema
  - Documentação de um sistema

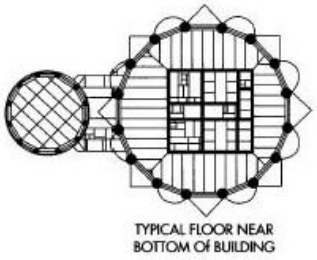


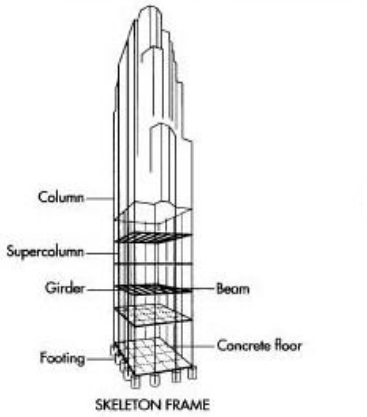
MODELAÇÃO DE UM SISTEMA

DEFINIÇÃO DOS PADRÕES DE ORGANIZAÇÃO DO SISTEMA









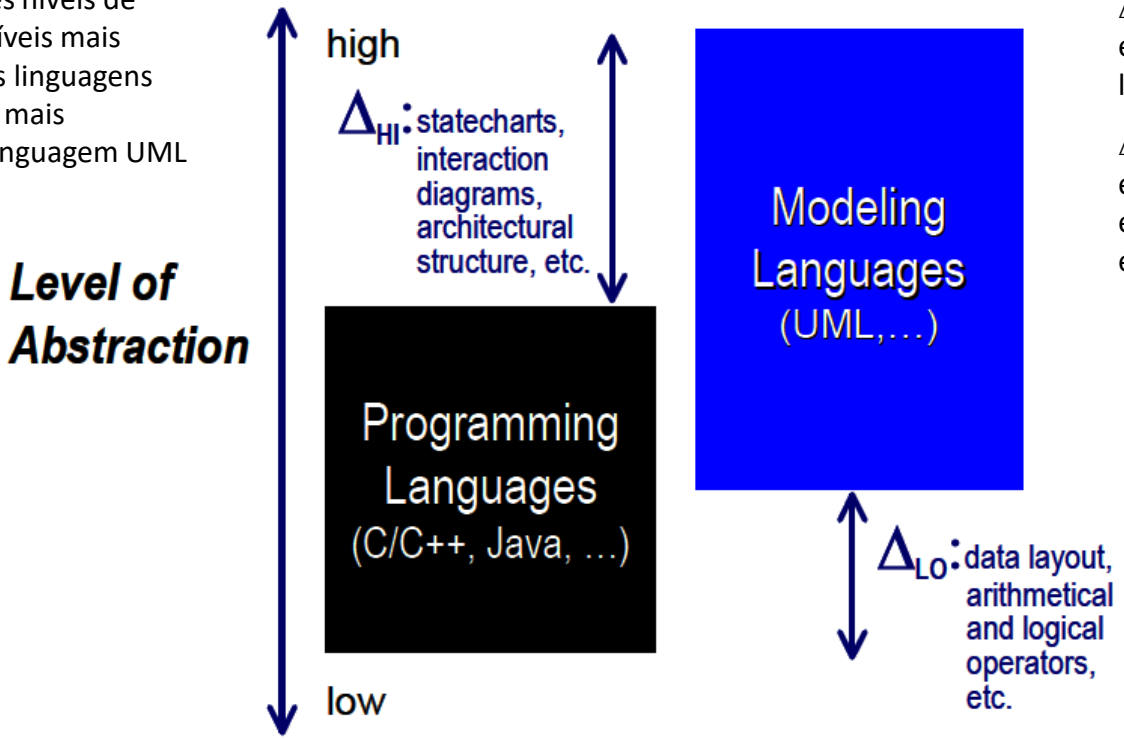
CARACTERÍSTICAS IMPORTANTES DE UM MODELO

- **ABSTRACÇÃO**
  - Foco nos aspectos importantes, remoção de aspectos não relevantes
- **COMPREENSÃO**
  - Facilidade de transmissão e compreensão das ideias envolvidas
- **PRECISÃO**
  - Representação correcta e rigorosa do sistema
- **PREVISÃO**
  - Possibilidade de inferência de conhecimento correcto acerca do sistema descrito

LINGUAGENS DE MODELAÇÃO

DESCRIÇÃO DO SISTEMA A DIFERENTES NÍVEIS DE ABSTRACÇÃO

As diferentes linguagens de especificação de software estão orientadas para a descrição de software a diferentes níveis de abstracção, desde níveis mais detalhados, como as linguagens *assembly*, até níveis mais abstractos como a linguagem UML



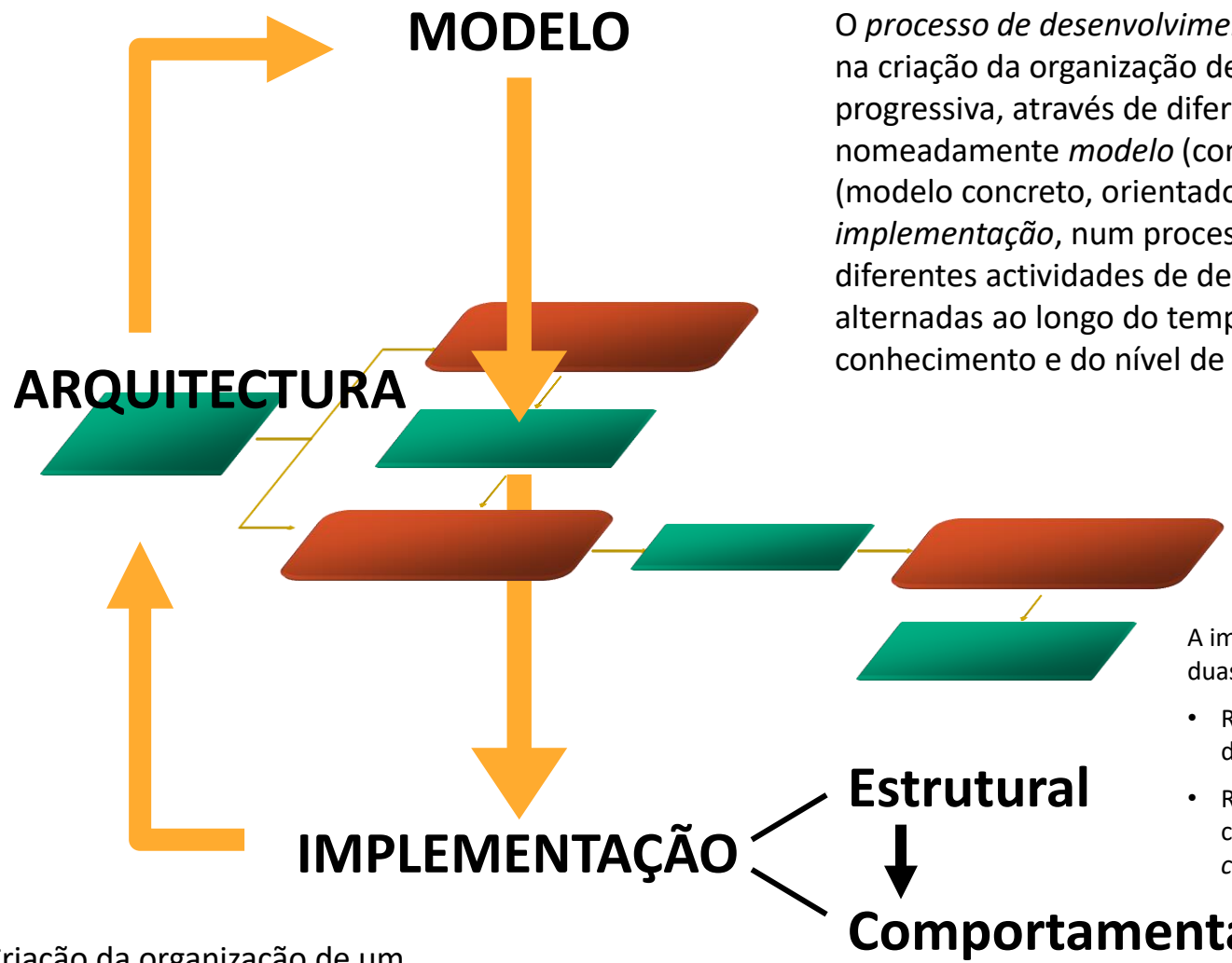
Elementos de descrição diferentes em cada nível

Δ_{HI} : Elementos disponíveis em UML não disponíveis em linguagens mais específicas

Δ_{LO} : Elementos disponíveis em linguagens mais específicas não disponíveis em UML

[Selic, 2003]

PROCESSO DE DESENVOLVIMENTO



O *processo de desenvolvimento* de software consiste na criação da organização de um sistema de forma progressiva, através de diferentes níveis de abstracção, nomeadamente *modelo* (conceptual), *arquitectura* (modelo concreto, orientado para a implementação) e *implementação*, num processo iterativo em que as diferentes actividades de desenvolvimento são alternadas ao longo do tempo em função do conhecimento e do nível de detalhe envolvido.

A implementação deve ser realizada em duas etapas principais:

- Realização de código relativo à estrutura do sistema (*código estrutural*)
- Realização de código relativo ao comportamento do sistema (*código comportamental*)

Criação da organização de um sistema de forma progressiva

PROCESSO ITERATIVO

BIBLIOGRAFIA

[Pressman, 2003]
R. Pressman, *Software Engineering: a Practitioner's Approach,* McGraw-Hill, 2003.

[Booch et al., 1998]
G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1998.

[Miles & Hamilton, 2006]
R. Miles, K. Hamilton, *Learning UML 2.0*, O'Reilly, 2006.

[Eriksson et al., 2004]
H. Eriksson, M. Penker, B. Lyons, D. Fado, *UML 2 Toolkit*, Wiley, 2004.

[Douglass, 2009]
B. Douglass, *Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development*, Addison-Wesley, 2009.

[SRC, 2015]
Semiconductor Research Corporation, *Rebooting the IT Revolution*, 2015.

[Korf, 1980]
R. Korf, Toward a model of representation changes, Artificial Intelligence, Volume 14, Issue 1, 1980.